# Towards Automated Test Case Generation Maturity

Urko Rueda
*Research Center on Software Production Methods*
*Universitat Politècnica de València*
Valencia, Spain
urueda@pros.upv.es

Fitsum Kifetew
*Fondazione Bruno Kessler*
Trento, Italy
kifetew@fbk.eu

Xavier Devroey
*Delft University of Technology*
Delft, The Netherlands
x.d.m.devroey@tudelft.nl

***Abstract*—This short paper reports our observations after six editions of the JUnitContest that benchmarks automated unit test generation tools for Java programs. We discuss our experience and depict the current state-of-the-art and identify potential future research directions. We advocate the use of benchmark as a standard practice to enhance maturity and foster adoption by the industry of automated test case generation tools.**

***Index Terms*—Maturation, Automation, Quality, Benchmarking, Challenges, Industry adoption**

## I. Introduction

Research in software testing has been going on for more than 6 decades and has demonstrated several advances and successes to enhance the software quality [1]. However, the transfer from research to industry has not always been a smooth process. In particular, despite some success stories [2], the automation of test case design is still far from industry expectations [3].

In other research fields, like hardware [4], information retrieval [5] or reverse engineering [6], benchmarking has demonstrated its utility to help standardization and maturation [7]. Benchmarking tools built by the research community on artifacts coming from industry (i) helps in promoting scientific maturity of a research field; (ii) helps researchers to understand problems encountered in the field, allowing them to improve their tools and identify new research directions.

Efforts to adopt benchmarking have been made by the software testing research community with works like Defects4J [8], the SF110 Corpus of Classes [9], JCrashPack,[1] TestBench, [2] TESTBEDS, [3] and the JUnitContest. [4] However, the absence of an established benchmark practice makes it hard to compare approaches [10]. In particular, the existence of benchmarks requires a community to support the maintenance effort and avoid deprecation of the benchmarks and the research they support. Also, the contributions must be evaluated against clearly defined standards to promote research that is collaborative, open and public.

[1]https://github.com/STAMP-project/JCrashPack
[2]https://personal.cis.strath.ac.uk/marc.roper/TestBench10/
[3]http://www.cs.umd.edu/~atif/testbeds/testbeds2011.htm
[4]https://github.com/PROSRESEARCHCENTER/junitcontest

## II. Observations from six editions of the JUnitContest

Reflecting on six years of benchmarking automated tools that generate JUnit tests for Java programs at the class level [11], we report some observations.

First, the JUnitContest benchmarking infrastructure is only suited for the Java programming language. The TIOBE [5] Programming Community index is an indicator of the popularity of programming languages. According to TIOBE Index for February 2019, Java is the most popular language, but we are only facing a small subset of code out in the market (Java representing 15.87%). There are yet 84.13% other programming languages for which benchmarking would be required to enable maturation of research prototype and, eventually, industry adoption.

Second, the JUnitContest has seen participation from a small number of automated unit test generation tools. However, results from the contest indicate that the combination (union) of the tests from all participating tools can outperform human developed tests. This suggests that existing tools tend to cover different areas of the testing problem, hence indicating the usefulness of considering a combined approach.

Third, open sourcing the JUnitContest infrastructure and competition results ease replication studies and comparison of other tools with state of the art. It allows comparison without requiring rerunning all the tools on the benchmark since the results are public as well. The statistics used to compare and rank the different tools also offer a reference standard.

Finally, the target benchmarks used in the last competitions have been built from open-source projects available online, ranging from simple to more complex and difficult, with respect to testing. Results of the contest also indicate the weaknesses of the existing tools, which is an important feedback for the research community. This feedback helps to identify the future research directions to pursue to improve the effectiveness of test generating tools.

## III. Challenges

Here are some challenges we identify from the previous editions of the JUnitContest competition.

[5]https://www.tiobe.com/tiobe-index/


https://doi.org/10.1109/SBST.2019.00011

### A. Programming languages landscape

The JUnitContest competition focuses on unit test generation for Java programs. However, applications are written in a large variety of languages and can even mix several ones. Research and benchmarking for other programming languages should take care of not reinventing the wheel. This calls for identification of commonalities and variabilities across programming languages and unit test generation tools to be able to identify cases where to apply specific testing techniques or solutions. And to dedicated unit test generation approaches working across programming languages boundaries.

### B. Collaborations for testing tools integration

As pointed out by previous results [11], integration of multiple unit test generation tools, each one relying on a specific approach, allows to achieve a higher effectiveness than using tools in isolation. Special focus should be dedicated to the integration of different approaches to achieve higher effectiveness without decreasing efficiency.

### C. High and low performing benchmarks

The benchmarks, coming from industrial open-source applications, and public results from the JUnitContest allow identifying the classes for which unit test generation is difficult. It offers an excellent opportunity for researchers to identify discrepancies between the current research state-of-the-art and industry state-of-the-practice. For instance, what characteristics hamper unit test generation via search-based or concolic-based approaches? Which generation technique is better, based on the characteristics of the class under test? etc.

## IV. Making benchmarking alive

This year we are celebrating the seventh edition of the JUnitContest, held at the SBST workshop co-located with ICSE'19. The source code of the contest infrastructure, and instructions on how to prepare a unit test generation tool to be used with this infrastructure, are publicly available[6]. Using the JUnitContest infrastructure offers several advantages, among which: experiments replication, comparison to other tools, standardization of the evaluation, etc. Furthermore, this year the contest infrastructure has been also made available as a Docker[7] image to facilitate easy setup and use. But ultimately, making benchmarking alive is in the hands of the community.

Since the first JUnitContest, crucial steps have been taken to foster maturation of automated unit test generation in a long journey for which, this position paper suggests some paths to follow.

## V. Discussion

In reality, the Software Testing community is focused on detecting/repairing faulty software developments. Should the research directions focus on repairing, or on helping develop better instead? Arguably, the load on the testing domain could be greatly reduced by employing formal code compilation (from models to code). Approaches are open to mitigate [12] by introducing formal methods for sound, correct and complete Software Production Processes. In the meantime, the Software Testing community is challenged to stay on the wave of modern software production (with its benefits and drawbacks), which - more than helping to guarantee the quality of software- makes it more difficult and complex to accomplish than ever (new devices, systems of systems, massive connectivity, Big Data applications, etc.).

[6]https://github.com/PROSRESEARCHCENTER/junitcontest

[7]https://www.docker.com

## Acknowledgment

This work has been partially supported by the Spanish State Research Agency and the Generalidad Valenciana under the projects DataME (TIN2016-80811-P). GISPRO (PROMETEO/2018/176), co-financed with ERDF, the EU Horizon 2020 ICT-10-2016-RIA "STAMP" project (No.731529), and the Dutch 4TU project "Big Software on the Run", and by the Italian Ministry of Education, University, and Research (MIUR) with the PRIN project GAUSS (grant n. 2015KWREMX).

## References

[1] D. Gelperin and B. Hetzel, "The growth of software testing," *Commun. ACM*, vol. 31, no. 6, pp. 687–695, Jun. 1988. [Online]. Available: http://doi.acm.org/10.1145/62959.62965

[2] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, "Deploying Search Based Software Engineering with Sapienz at Facebook," in *Search-Based Software Engineering. SSBSE 2018.*, ser. LNCS, vol. 11036. Springer, 2018.

[3] V. Garousi and M. V. Mäntylä, "When and what to automate in software testing? a multi-vocal literature review," *Information & Software Technology*, vol. 76, pp. 92–117, 2016.

[4] J. L. Henning, "Spec cpu2000: measuring cpu performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, July 2000.

[5] A. Rorissa, "Image retrieval: Benchmarking visual information indexing and retrieval systems," *Bulletin of the American Society for Information Science and Technology*, vol. 33, no. 3, pp. 15–17, 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/bult.2007.BULT1720330310

[6] S. E. Sim, R. C. Holt, and S. Easterbrook, "On using a benchmark to evaluate c++ extractors," in *Proceedings 10th International Workshop on Program Comprehension*, June 2002, pp. 114–123.

[7] S. E. Sim, S. Easterbrook, and R. C. Holt, "Using benchmarking to advance research: A challenge to software engineering," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 74–83. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776826

[8] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: a database of existing faults to enable controlled testing studies for Java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis - ISSTA 2014*. ACM Press, 2014, pp. 437–440.

[9] G. Fraser and A. Arcuri, "A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 2, pp. 1–42, dec 2014.

[10] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4, pp. 531–577, Aug 2012. [Online]. Available: https://doi.org/10.1007/s10664-011-9173-9

[11] U. Rueda Molina, F. Kifetew, and A. Panichella, "Java unit testing tool competition: Sixth round," in *Proceedings of the 11th International Workshop on Search-Based Software Testing*, ser. SBST '18. New York, NY, USA: ACM, 2018, pp. 22–29. [Online]. Available: http://doi.acm.org/10.1145/3194718.3194728

[12] O. Pastor and V. Pelechano, *The Conceptual Model Is The Code. Why Not?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 153–159.